# Barnes-Hut algoritam

(problem n tijela)

Sandro Lovnički, Oblikovanje i Analiza Algoritama
PMF - Matematika

# Što nas zanima?

- promatramo n tijela (zadanih svojim koordinatama i svojstvom ključnim za silu) koja nekom silom (gravitacijskom, elektomagnetskom,...) djeluju jedno na drugo
- htjeli bismo što efikasnije izračunati rezultantnu silu koja djeluje na svako pojedino tijelo

# Gdje se skriva problem?

- u tipičnim sustavima koje želimo promatrati nalazi se $n>>1$ tijela
- računanje sila kojima svako od preostalih n-1 tijela djeluje na promatrano, i tako za sva tijela, složenosti je $O(n^2)$

# Ideja

- u praktičnim fizikalnim primjerima, nije nam bitna savršena točnost
- dovoljno bliska tijela aproksimiramo njihovim centrom mase (slično za sile koje ne ovise o masi):

$$m = \Sigma_i m_i \qquad (f1)$$

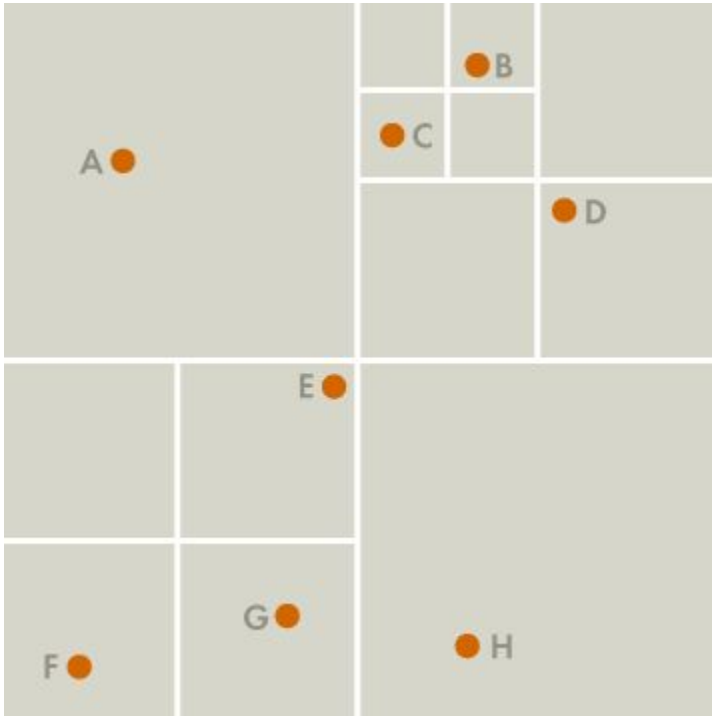
$$x = \Sigma_i x_i * m_i \; / \; m \qquad (f2)$$

$$y = \Sigma_i y_i * m_i \; / \; m \qquad (f3)$$

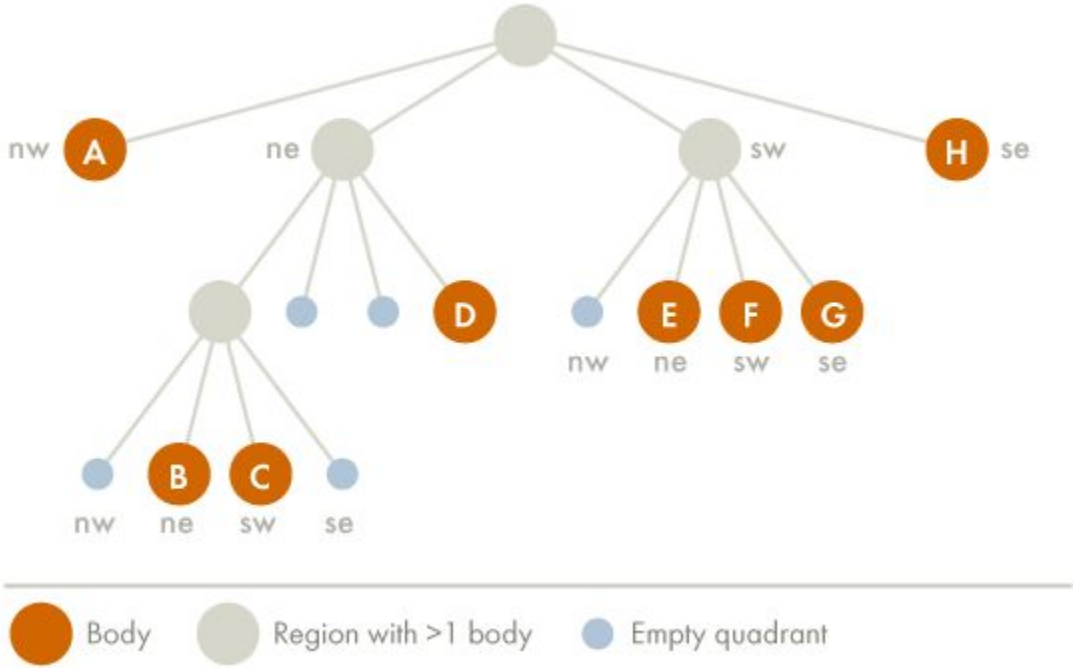
# Ideja

- jasno, trebamo i računanje centara brže od $O(n^2)$
- tijela spremamo u oktalno (za 3D) ili kvartarno (za 2D) stablo tako da svako od n tijela bude u različitom listu te su dopušteni prazni listovi
- nakon umetanja, obilaskom stabla računamo centre i sile koje djeluju na tijela
- *(sve će postat jasnije na idućem slajdu)*

# Ideja




s1: tijela u prostoru




s2: tijela u kvartarnom stablu

http://arborjs.org/docs/barnes-hut

# Kreiranje stabla (pseudo)

1. tijelo ubacujemo u čvor ako je list i nema ničega u njemu
2. ako je bio list *(dakle neprazan)*, kreiramo mu 4 djece te prebacimo njegov sadržaj u odgovarajuće dijete ovim istim algoritmom *(prolazi prvi korak)*
3. rekurzivno zovemo algoritam na djeci

# Kreiranje stabla (kod)

```
void quadTree::insert(Data* d) {
    if(centerMass.mass == 0.0 && NE == nullptr) {
        centerMass = *d;
        return;
    }
    //if leaf, create subtrees
    if(NE == nullptr) {
        NE = new quadTree(this,Rectangle(Point(area.center.x-area.diameter/2,area.center.y+area.diameter/2),area.diameter/2));
        NW = new quadTree(this,Rectangle(Point(area.center.x+area.diameter/2,area.center.y+area.diameter/2),area.diameter/2));
        SE = new quadTree(this,Rectangle(Point(area.center.x-area.diameter/2,area.center.y-area.diameter/2),area.diameter/2));
        SW = new quadTree(this,Rectangle(Point(area.center.x+area.diameter/2,area.center.y-area.diameter/2),area.diameter/2));
        //move old data
        if(centerMass.coordinates.x <= area.center.x && centerMass.coordinates.y >= area.center.y)
            NE->insert(new Data(Point(centerMass.coordinates.x,centerMass.coordinates.y),Point(centerMass.force.x,centerMass.force.y),centerMass.mass));
        else  if(centerMass.coordinates.x >= area.center.x && centerMass.coordinates.y >= area.center.y)
            NW->insert(new Data(Point(centerMass.coordinates.x,centerMass.coordinates.y),Point(centerMass.force.x,centerMass.force.y),centerMass.mass));
        else if(centerMass.coordinates.x <= area.center.x && centerMass.coordinates.y <= area.center.y)
            SE->insert(new Data(Point(centerMass.coordinates.x,centerMass.coordinates.y),Point(centerMass.force.x,centerMass.force.y),centerMass.mass));
        else if(centerMass.coordinates.x >= area.center.x && centerMass.coordinates.y <= area.center.y)
            SW->insert(new Data(Point(centerMass.coordinates.x,centerMass.coordinates.y),Point(centerMass.force.x,centerMass.force.y),centerMass.mass));
        centerMass.mass = 0.0;
    }

    //insert into children
    if(d->coordinates.x <= area.center.x && d->coordinates.y >= area.center.y)
        NE->insert(d);
    else if(d->coordinates.x >= area.center.x && d->coordinates.y >= area.center.y)
        NW->insert(d);
    else if(d->coordinates.x <= area.center.x && d->coordinates.y <= area.center.y)
        SE->insert(d);
    else if(d->coordinates.x >= area.center.x && d->coordinates.y <= area.center.y)
        SW->insert(d);
}
```

# Računanje centara masa (pseudo)

1. ako je čvor list, već ima centar mase (masa i koordinate tijela u njemu ili 0)
2. ako nije list, prvo rekurzivno ovim algoritmom izračunamo centre masa njegovih 4 djece pa njegov po formulama (f1)-(f3) za i=0,1,2,3.

# Računanje centara masa (kod)

```
void quadTree::calculateCenterMass() {
    //if leaf, already has centerMass
    if(NE == nullptr)
        return;
    //calculate children first
    double x,y,m;
    NE->calculateCenterMass();
    NW->calculateCenterMass();
    SE->calculateCenterMass();
    SW->calculateCenterMass();
    //calculate
    m = NE->centerMass.mass + NW->centerMass.mass + SE->centerMass.mass + SW->centerMass.mass;
    x = NE->centerMass.coordinates.x*NE->centerMass.mass + NW->centerMass.coordinates.x*NW->centerMass.mass +
            SE->centerMass.coordinates.x*SE->centerMass.mass + SW->centerMass.coordinates.x*SW->centerMass.mass;
    y = NE->centerMass.coordinates.y*NE->centerMass.mass + NW->centerMass.coordinates.y*NW->centerMass.mass +
            SE->centerMass.coordinates.y*SE->centerMass.mass + SW->centerMass.coordinates.y*SW->centerMass.mass;
    centerMass.coordinates.x = x/m;
    centerMass.coordinates.y = y/m;
    centerMass.mass = m;
}
```

# Računanje sile (pseudo)

1. ako smo u praznom listu, ne radimo ništa
2. ako smo u nepraznom listu koje nije naše tijelo za koje računamo, izravno izračunamo silu između tog tijela i našeg
3. ako nismo u listu i taj centar mase je dovoljno udaljen od našeg tijela, izračunamo silu između njega i našeg tijela
4. rekurzivno prolazimo po djeci

# Računanje sile (kod)

```
void quadTree::calculateForce(Data* d) {
    //nothing in node (empty leaf)
    if(centerMass.mass == 0.0) return;

    double deltaX = centerMass.coordinates.x - d->coordinates.x;
    double deltaY = centerMass.coordinates.y - d->coordinates.y;
    double distance = sqrt(pow(deltaX,2) + pow(deltaY,2));

    //does not exert force on itself
    if(distance == 0) return;
    //if too close, "gets eaten"
    if(distance < 8) {
        d->coordinates.x = centerMass.coordinates.x;
        d->coordinates.y = centerMass.coordinates.y;
        return;
    }

    //for bodies, calculate directly
    if(NE == nullptr) {
        d->force.x += P * (deltaX / distance) * (d->mass * centerMass.mass / pow(distance,2));
        d->force.y += P * (deltaY / distance) * (d->mass * centerMass.mass / pow(distance,2));
        return;
    }
    //for sufficiently far centerMasses
    if(area.diameter / distance < 0.5) {
        d->force.x += P * (deltaX / distance) * (d->mass * centerMass.mass / pow(distance,2));
        d->force.y += P * (deltaY / distance) * (d->mass * centerMass.mass / pow(distance,2));
        return;
    }
    //all exist because first 'if' failed
    NE->calculateForce(d);
    NW->calculateForce(d);
    SE->calculateForce(d);
    SW->calculateForce(d);
}
```

# Računanje kretanja

- diskretizirano;

$$s = t^2 * F/m \qquad (f4)$$

```
p0->coordinates.x += pow(t,2) * p0->force.x / p0->mass;
p0->coordinates.y += pow(t,2) * p0->force.y / p0->mass;
```

# Rezultati

- dobili smo izrazito zadovoljavajuć algoritam složenosti O(nlogn) što predstavlja ogromnu uštedu u “near real-time” modeliranju sustava velikih razmjera

- broj računanja sila tijekom izvršavanja algoritma: između 21 i 26 tijekom “sredine radnje” s podjednakim ulaskom u čvorove s tijelima i prijevremenim računanjem iz centra mase

# Zanimljivosti i preinake

- sama implementacija
- spriječavanje stvaranja nepotrebno dubokog stabla
- namještanje “gravitacijske konstante” i vremenskog koraka
- borba s prevelikim silama

# Literatura


- *http://arborjs.org/docs/barnes-hut*
  (pristupljeno 26.1.2017.)